Assignment 3

This assignment has three parts: two small problems in physics and one extension of the planning problem for arms.

1. A state trooper spies a speeding car going at a speed of 100 miles/hour on I-95. The trooper's car can accelerate as much as 12 miles/hour/second.
   - Write a simulator and GUI for this problem. Your GUI should have "Go" and "Quit" buttons, should depict the two cars going across from left to right and should show the point the two cars meet. Your simulator should find the distance traveled for each car as a function of time.
   - At what time and distance does the trooper catch up with the speeding car?
   - What is the trooper's speed at the moment they catch up?

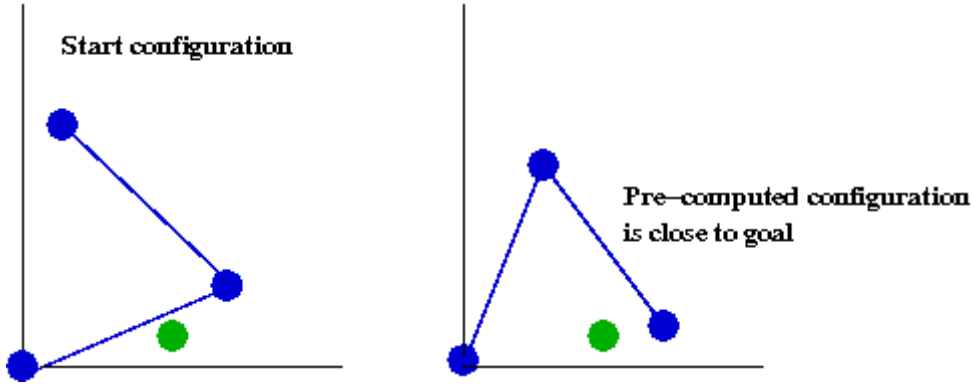
   Note: if you haven't written any GUI code before, a good starting point is the "moving object" code from Module 3 (Exercise 37).

2. Consider the motion of a skydiver, both in freefall and with a chute deployed. To make this more realistic, we will model *drag*. Generally, the higher the velocity, the higher the drag. Thus, a simple model for the downward acceleration is:

   $a(t) = g - k\,v(t)$

   where k is a constant that depends on the air-resistance offered by the skydiver. Suppose the skydiver pops the chute at time $t=T$. Then, we will assume $k=0.268$ (low drag) when $t<T$ (before the chute opens) and $k=1.342$ (high drag) after the chute opens.
   - Write a simulator for this problem and a simple GUI depicting the skydiver - make sure that something changes on screen to differentiate between freefall and when the chute is open. Your GUI should have a "Go" button and a "Quit" button. You can, of course, make your GUI elaborate (use an image of a skydiver etc).
   - Suppose that the skydiver is able to land at a velocity of 40 feet/sec. When should the chute be opened at the latest assuming the fall starts at a height of 1200 feet?
   - Plot the velocity function for the skydiver.

3. As you saw in class, even the A* planner takes a long time for some goal-configurations in the arm problem. In this assignment, you will explore the idea of storing pre-computed plans and using them. The idea is this: if we are able to pre-compute and store k plans, then the end-configurations of those plans can each be used as starting points for an altogether new goal. The thinking is, perhaps one of those end-configurations is close to the goal, in which case a plan can be generated quickly by combining the pre-computed plan with the plan that starts with the end-configuration of the pre-computed plan.



   In the picture above, we are given a start configuration (left) and a goal (in green). If we had already pre-computed a few plans, one of them could be the configuration at the right. This means we know how to get

to that configuration quickly from the start configuration. Thus, in a new problem, we can *start* the search from there, compute the steps to the goal node, and add these steps to the pre-computed plan.

- Modify the A* algorithm to compute *k=5* stored plans. Naturally, you ought to pick these plans carefully so that they, in a sense, "cover" the range of difficult situations.
- Then select a few new targets and compare regular A* with your new algorithm, which we'll call pre-A*. How will you choose which pre-stored plan to use? Alternatively, you could run all 5 of them (by stepping through each in turn). How often does pre-A* outperform A*?
- [Optional for undergrads] Experiment with *k* and implement a parallel search, in which you treat each pre-stored plan as a separate search from that starting point, stepping through each in turn. At what values of *k* does the parallel-search become worse than the single run of A*? To make a fair comparison, add up the total time for a number of different goal states.
- [Optional for undergrads] Use an arm with 4 links instead of 3. How does this change your findings? You can change the number of links by setting `numLinks=4` in `ArmProblem.java`.

Submission:

- Put all your code for this assignment in a single directory so that your code unpacks into that directory. Embed your username in the directory name.
- Include the full planning package from class so that we're able to upload your algorithm into `PlanningGUI`.
- Include a plain-text README file that tells us how to run your code for each of the three problems. The README file should also have numerical answers and results pertaining to the three problems.
- You can make your GUI's look nice (out of pride), but there won't be very many points for appearance, so don't expect huge points for spending time on this.
- Name zip file `karel3.zip` (for username `karel`).
- Submit your zip file using Blackboard.
